

VPN, Online-Services

Wireguard, Tunneling, SSH,

- [Bitwarden im eigenen VPN](#)
- [Wireguard Client](#)
- [Wireguard Server Installation ganz einfach!](#)
- [DS-Lite IPv6 Tunnel - Port Forwarding - NAT - WireGuard VPN](#)
- [IPTABLES](#)

Bitwarden im eigenen VPN

Hallo

ich habe vor einiger Zeit Bitwarden/Vaultwarden für mich entdeckt und wollte dies **nur** über meinen internen VPN ohne Portfreigabe benutzen. Leider, ist dies Zwecks https Zwang gar nicht so einfach. Leider findet man im Internet nicht wirklich gute bzw. einfache Lösungen. Nach stundenlanger Recherche und rum probieren habe ich mit Hilfe eines netten Discord Users, die für mich perfekte Lösung gefunden und würde diese gerne mit euch teilen

Ich gehe davon aus das bei allen Docker installiert ist:

Falls ihr OMV benutzt müsst ihr noch den Port 80 auf einen anderen Port umlegen, da nginx den Port 80 benötigt.

1. nginx installieren

Als erstes installieren wir nginx in unserem Docker, ich habe dies über Stacks getan, hier mal der Auszug mit Erklärungen

Code:

```
version: '3'
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    restart: unless-stopped
    ports:
      - '80:80' #zwingend Port 80 für http Seiten
      - '511:81'
      - '443:443'#zwingend Port 443 für https Seiten
    volumes:
      - eurerSpeicherpfad:/data
      - eurerSpeicherpfad:/etc/letsencrypt
```

Ihr könnt natürlich auch einen anderen Port benutzen, dann müsst ihr bei eurer gewünschten Seite den Port mit angeben (*zum Beispiel paddy0293.local.de:6555*), was eigentlich nicht Sinn der Sache ist.

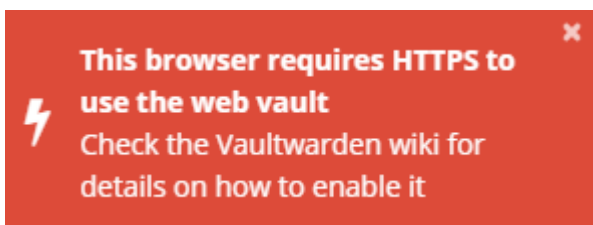
2. vaultwarden installieren

Code:

```
version: '3'

services:
  vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: always
    ports:
      3000:80 # könnt 3000, wie euch beliebt
    environment:
      - WEBSOCKET_ENABLED=true # Enable WebSocket notifications.
      - ADMIN_TOKEN={{ADMIN_TOKEN}} # Passwort setzen für das Admin Panel
      - SIGNUPS_ALLOWED=false
    volumes:
      - ./vw-data:/data
```

Jetzt kommt das Problem weshalb ich den ganzen Aufwand überhaupt betrieben habe:



Jetzt gibt es 2 Varianten, eine kostenlose und eine kostenpflichtige (2,50€ im Jahr)

Ich erläutere erst einmal nur die kostenpflichtige Variante wenn Bedarf besteht auch gerne die kostenlose

Ich habe mir eine Domain gekauft über netcup gekauft, über die ich meine Bitwarden Domain ansteuern möchte.

Ich kann ab hier nur noch von Netcup sprechen.

Sobald ihr die Domain gekauft habt müsst ihr euch eine API erstellen, die ist zwingend notwendig für das Letsencrypt Zertifikat.

Die macht ihr im Netcup Controlpanel unter Stammdaten -> API

Nun erstellt ihr ein API Passwort und ein API Key

Stammdaten

> API-Verwaltung

Übersicht

Wenn Sie unsere Schnittstelle verwenden, können Sie hier die nötigen Einstellungen machen.
Um auf dem API Domainfunktionen zu nutzen, schließen sie bitte einen Domainresellervertrag mit uns.

[API-Passwort generieren](#)

☰ API-Keys



[API-Key löschen](#)

Jetzt lassen wir via nginx ein letsencrypt Zertifikat erstellen

Add Let's Encrypt Certificate



Domain Names *

These domains must be already configured to point to this installation

Email Address for Let's Encrypt *

Use a DNS Challenge

This section requires some knowledge about Certbot and its DNS plugins. Please consult the respective plugins documentation.

DNS Provider *

Credentials File Content *

```
dns_netcup_customer_id = 123456  
dns_netcup_api_key = 0123456789abcdef0123456789abcdef01234567
```

This plugin requires a configuration file containing an API token or other credentials to your provider

This data will be stored as plaintext in the database and in a file!

Propagation Seconds

Leave empty to use the plugins default value. Number of seconds to wait for DNS propagation.

I Agree to the [Let's Encrypt Terms of Service](#) *

Cancel

Save

dns_netcup_customer_id = eure Login ID
dns_netcup_api_key = euren Netcup API Key

dns_netcup_api_password = euer Netcup API Password
Propagation Seconds = 600 #wird von netcup empfohlen

Dann auf Save und einen Kaffee trinken gehen, dies dauert einen kleinen Moment

So jetzt haben wir erfolgreich schon mal unser Cert erstellt ohne das wir einen Port öffnen mussten
Jetzt müssen wir nur noch ein Proxhost erstellen.

The screenshot shows the 'New Proxy Host' configuration form in the Netcup interface. The navigation bar at the top includes 'Dashboard', 'Hosts', 'Access Lists', 'SSL Certificates', 'Users', 'Audit Log', and 'Settings'. The 'Hosts' menu is open, highlighting 'Proxy Hosts'. The form itself has a title 'New Proxy Host' and a close button. Below the title are tabs for 'Details', 'Custom locations', 'SSL', and 'Advanced'. The 'Details' tab is active. The form contains the following fields and options:

- Domain Names ***: A text input field containing 'eurenetcupdomain.de'.
- Scheme ***: A dropdown menu with 'http' selected.
- Forward Hostname / IP ***: A text input field containing 'eure vaultwardenip'.
- Forward Port ***: A dropdown menu with '80' selected.
- Cache Assets**: A toggle switch, currently turned off.
- Block Common Exploits**: A toggle switch, currently turned off.
- Websockets Support**: A toggle switch, currently turned off.
- Access List**: A text input field containing 'Publicly Accessible'.

At the bottom right of the form are two buttons: 'Cancel' and 'Save'.

Unter SSL, euer grad erstelltes Cert auswählen

New Proxy Host



↶ Details Custom locations SSL Advanced

SSL Certificate



Force SSL HTTP/2 Support
 HSTS Enabled HSTS Subdomains

Cancel

Save

Speichern, schon habt ihr eine erfolgreiche Umleitung auf euren Vault via Hhttps.

Jetzt kommt es darauf an, ob ihr über einen DNS Server verfügt oder nicht.

Wenn ja, dann bitte die Domain mit eurer IP im DNS verknüpfen.

Wenn Nein, dann bleibt euch nur die Möglichkeit die Domain über die Hostdatei einzufügen, diese findet ihr unter:

System32\drivers\etc\hosts

Diese Datei editieren und am Zeilenende folgendes schreiben.

eure Valtwarden IP und eure Domain

Bsp. 192.168.0.0 netcupdomain.de

Speichern, in cmd noch ipconfig /flushdns -> löscht den Cache

Eure Netcup Domain ansurfen und glücklich sein

Wenn ihr via Handy draufzugreifen wollt müsst ihr nur eure lokale IP eingeben, diese benötigt keine Hhttps und somit müsst ihr auch keine Host Datei anpassen.

Ich hoffe das kleine Tutorial hilft euch

Bei Fragen oder Feedback einfach hier melden.

Viele Grüße

Paddy

Wireguard Client

WIKI: [Wiki Debian](#)

Listing:

```
apt-get install openresolv
```

```
ip link add wg type wireguard
```

```
wg show
```

```
nano /etc/wireguard/wg.conf
```

```
wg-quick up wg
```

```
ip address
```

```
ping 10.8.0.2
```

```
wg show
```

- Service:
- systemctl enable [wg-quick@wg](#)
- systemctl daemon-reload
- systemctl start [wg-quick@wg](#)

```
=>systemctl list-units|grep wg
```

Routing:

```
sysctl net.ipv4.ip_forward=1
```

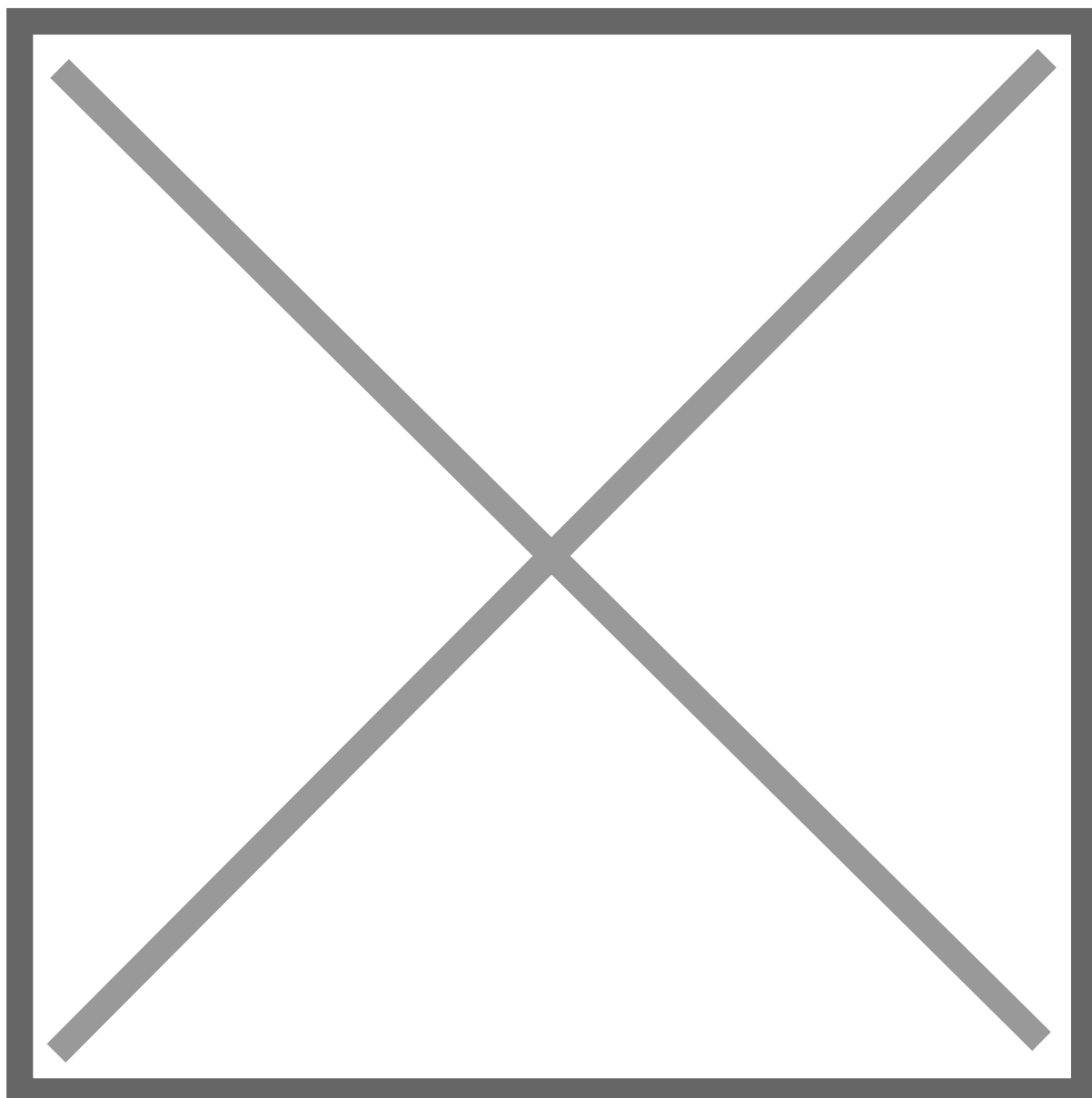
WireGuard Site to Site VPN –

Zwei Netzwerke sicher

verbinden

WireGuard ist der "neue" VPN Server von Heute! Nicht das er als Open Source sowieso schon bei mir gewonnen hat, sondern er ist auch ein Leichtgewicht. Der schlanke SourceCode bringt ihm zusätzlich Performance, sodass wir alles an Board haben was wir brauchen.

Zusammengefasst, WireGuard ist ein sehr **einfacher**, sehr **sicherer** und sehr **schneller** VPN Server für alle möglichen **Plattformen** und **Betriebssystemen**.



Im letzten Video habe ich euch gezeigt wie man eine Client to Site Verbindung herstellt und habe euch die Konfiguration dazu hier im Blog aufgeschlüsselt. ([WireGuard Client VPN](#))
Jetzt zeige ich euch wie man eine Site-to-Site VPN Verbindung herstellt und dann die Netzwerke auf der gegenüberliegenden Seite erreicht

Anleitung: WireGuard Site-to-Site VPN Server

Schritt 1: Auf dem SERVER1 und SERVER2 ausführen

?

```
1 sh -c "echo 'deb http://deb.debian.org/debian buster-  
2 backports main contrib non-free' >  
3 /etc/apt/sources.list.d/buster-backports.list"  
4 apt update  
apt install linux-headers-$(uname --kernel-release)  
apt install wireguard
```

?

```
1 nano /etc/sysctl .conf  
2 # <- entfernen bei der Zeile "net.ipv4.ip_forward=1" und  
3 speichern  
sysctl -p
```

?

```
1 cd /etc/wireguard/  
2 umask 077; wg genkey | tee privatekey | wg pubkey >  
3 publickey  
4 cat privatekey  
5 cat publickey  
chmod 600 /etc/wireguard/privatekey
```

Schritt 2: **Nur** auf SERVER1 ausführen

Konfigurationsdatei anlegen

nano /etc/wireguard/wg0.conf

```
[Interface]  
PrivateKey = <Dein privater Schlüssel [privatekey SERVER 1]>  
Address = 172.31.0.1/24  
SaveConfig = true  
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUEF  
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQ
```

```
ListenPort = 51820
```

```
[Peer]
```

```
PublicKey = <Öffentlicher Schlüssel von [SERVER 2]>
```

```
AllowedIPs = 192.168.0.0/24, 172.31.0.0/24
```

```
PersistentKeepalive = 25
```

In den Zeilen **PostUP & PostDown** das Interface anpassen wie es zu eurem System passt. Hier steht jetzt **eth0**, das kann aber bei euch anders sein. Prüfen könnt Ihr das mit **“ip a”**

Schritt 3: Nur auf SERVER2 ausführen

Konfigurationsdatei anlegen

nano /etc/wireguard/wg0.conf

```
[Interface]
```

```
PrivateKey = <Dein privater Schlüssel [privatekey SERVER 2]>
```

```
Address = 172.31.0.2/24
```

```
SaveConfig = true
```

```
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

```
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
```

```
[Peer]
```

```
PublicKey = <Öffentlicher Schlüssel von [SERVER 1]>
```

```
Endpoint = <ServerIP oder Domain/FQDN>:51820
```

```
AllowedIPs = 10.0.0.0/8, 172.31.0.0/24
```

```
PersistentKeepalive = 25
```

Schritt 4: Auf SERVER1 und SERVER2 ausführen

?

```
1  wg-quick up wg0
2  wg
3  ##Das zeigt dir den Status der Verbindung an
```

Das **wg-quick** Kommando startet die Wireguard Verbindung mit der entsprechend hinterlegten Konfiguration, welche wir im vorherigen Schritt erstellt haben. Wenn eure Konfigurationsdatei anders heißen sollte, müsst Ihr den Befehl natürlich anpassen. **wg0** ist hier nur als Beispiel genommen.

Schritt 5: Autostart (Server1 & Server2)

Wenn **WireGuard** automatisch mit dem System starten soll, dann folgenden Befehl absetzen.

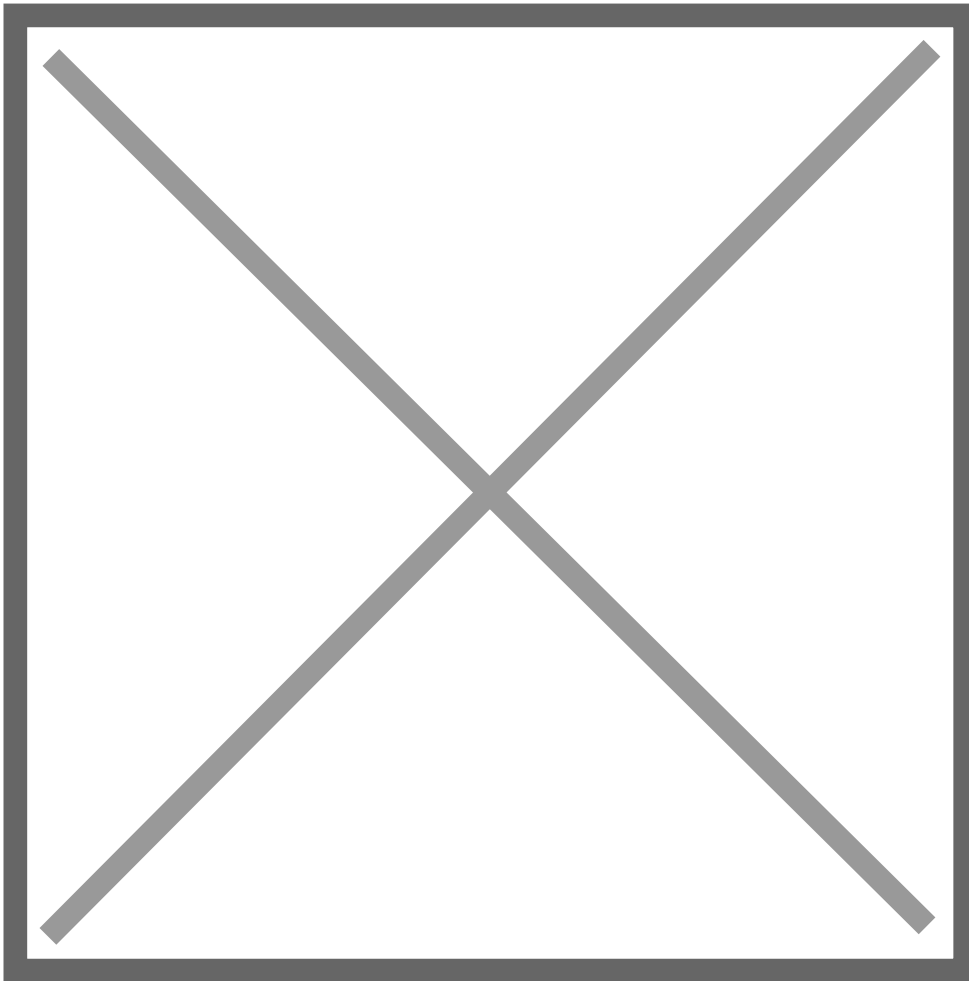
?

1

systemctl enable wg-quick@wg0

Schritt 6: Routing

Ihr müsst natürlich auf beiden Seiten für das Routing sorgen. Hier ein Auszug aus dem Video wie das Routing in der Fritzbox aussehen könnte.



Fritzbox Routing

Wireguard Netzwerk mit Routing einrichten

[28. November 2019](#) by [Sascha Pfau](#) · [Keine Kommentare](#)

Wireguard Netzwerk

Die prinzipielle Einrichtung von Wireguard habe ich in dem Artikel [Wireguard](#) beschrieben. Es ging um die die Point to Point Einrichtung mehrerer Clients die auf einen PC, der als Server fungiert, zugreifen. Die Clients konnten sich untereinander nicht sehen.

Im folgenden möchte ich beschreiben, wie man ein VPN Netz aufbaut in dem sich die Clients untereinander verbinden können. Damit Clients in dem privaten Netzwerk sich über Hostname ansprechen können, wird ein zentraler DNS Server eingerichtet. Hierfür verwende ich einen vorhandenen OpenWRT Router.

Aufbau des Netzwerkes

Das Netzwerk ist auf mehrere Standorte verteilt. Es gibt einen zentralen Linux PC der die Rolle des Wireguard Servers einnimmt. Die Rolle einnimmt, weil es per se keinen Wireguard Server gibt. Es ist die Konfiguration die ihn zur Zentrale werden lässt. Damit dieser erreichbar ist, ist ein DynDNS Dienst oder eine feste IP Adresse von Nöten. Dieses ist aber nicht Bestandteil dieses Artikels. Hier verweise ich zum Beispiel auf den Artikel [DynDNS mit Duck DNS](#).

Der Server bekommt in dem VPN die Adresse 10.0.0.1. Der Server ist für das Routing unter den Clients verantwortlich. Das heißt er leitet eingehende Pakete an die anderen Clients weiter. Hierfür muss das IP Forwarding aktiviert und einfache Regeln für die **FORWARD** Chain von IPTABLES angelegt werden.

Die Clients stellen eine Reihe von Laptops, Workstations und weiteren Servern da. Die Client erhalten durchnummerierte IP Adressen beginnend mit 10.0.0.2/32. In meinem Setup sind es aktuell 6 weitere PCs die in dem VPN vorhanden sind.

Konfiguration von Wireguard

Im Folgenden werde ich das Schema wie Clients und der Server einzurichten sind erläutern. Wenn das Schema bekannt ist, dann ist es leicht weitere Clients hinzuzufügen.

Einrichtung auf dem Server

Das Schema ist eigentlich recht einfach. Es muss allerdings auf die Subnetzmaske bei der Angabe der IP Adressen genau geachtet werden. Die Server Adresse ist mit /24 und die Clients mit /32 anzugeben.

```
[Interface]
PrivateKey = PrivateKeyDesServers
ListenPort = 32768
Address = 10.0.0.1/24

# Allow routing between clients
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT

#
# Client 1
#
[Peer]
PublicKey = PublicKeyDesClient
AllowedIPs = 10.0.0.2/32

#
# Client 2
#
[Peer]
PublicKey = PublicKeyDesClient
AllowedIPs = 10.0.0.3/32

#
# Client 3
#
[Peer]
PublicKey = PublicKeyDesClient
AllowedIPs = 10.0.0.4/32
```

```
.  
. .  
  
#  
# Client 7  
#  
[Peer]  
PublicKey = PublicKeyDesClient  
AllowedIPs = 10.0.0.7/32
```

Einrichtung der Clients

Es ist hierbei wichtig zu wissen, dass nur ein Client das gesamte Subnetz beanspruchen darf! Sonst funktioniert das ganze nicht. Das heißt auf den Clients sind explizit die Adresse einzutragen mit denen der Client kommunizieren darf. Dieses macht aber nur bei kleineren Netzen einen Sinn.

```
[Interface]  
PrivateKey = PrivateKeyDesClients  
Address = 10.0.0.3/24  
DNS = 10.0.0.7  
  
[Peer]  
PublicKey = PublicKeyDesServers  
AllowedIPs = 10.0.0.1/32, 10.0.0.2/32, 10.0.0.5/32, 10.0.0.6/32, 10.0.0.7/32  
EndPoint = myname.dyndns.org:32222  
PersistentKeepalive = 25
```

Wichtig ist das alle Clients mit **10.0.0.X/32** angegeben werden.

IP Forwarding Aktivierung

Das allgemeine IP Forwarding bzw. auf Geräteebene aktivieren habe ich bereits in dem [Wireguard](#) Artikel beschreiben.

Firewall Regeln

Damit die Pakete nun auch an die Client weitergeleitet werden, muss in der **FORWARD** Chain das In-/Output auf dem Wireguard Interface zulassen. Dieses kann mit dem **PostUp** in der Konfiguration automatisch nach dem Starten der Verbindung durchgeführt werden.

```
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT
```

Verbindungsaufbau

Für den automatischen Verbindungsaufbau muss die Option **PersistentKeepalive** gesetzt werden, damit ein Paket Verbindung öffnet.

Einrichtung von DNS

Auf dem Client habe ich die Konfiguration mit dem KDE NetworkManager durchgeführt. Bei der Wireguard Konfiguration habe ich nun die Adresse des DNS Servers eingetragen, der für das interne Netz zuständig ist. Die [Konfiguration der statische Hostnames](#) beschreibt ein andere Artikel.

Testen der Namesauflösung

Auf <http://openwrt/cgi-bin/luci/admin/network/dhcp> unter *Local Server* mit **/XXXX.duckdns.org/** Anfrage an die Domain nach außen unterbinden, da diese sonst von DuckDNS beantwortet werden würden und somit die IPV6 Adresse genommen wird, was zu Fehlern führt.

[Wireguard](#)

Wireguard Server Installation ganz einfach!

Wireguard Server Installation ganz einfach!

Wer einen eigenen Wireguard Server betreiben möchte steht meistens vor der Herausforderung einige Fachkenntnisse mitzubringen. Daher haben wir eine Lösung entwickelt mit der es ganz einfach auch für Anfänger wird. In unserer Anleitung zeigen wir Dir den Weg und beschreiben jedes Detail des Vorganges. Mit dieser Anleitung sollten daher auch Neueinsteiger zu Recht kommen und in wenigen Minuten Ihren eigenen Wireguard Server in Betrieb haben.

Inhaltsverzeichnis

Der eigene Wireguard VPN Server.

In dieser Anleitung zeigen wir wie man einen VPN Server mit Wireguard selbst aufsetzt, dabei einen virtuellen Cloud Server verwendet mit einer statischen IP Adresse und diesen für verschiedene Anwendungen verwenden kann. Das ganze basiert auf dem Betriebssystem Ubuntu 20.04.xx und der Software und die Codebasis, welche Jason A. Donenfeld (Entwickler von Wireguard) veröffentlicht hat. Es ist aber auch möglich, allerdings nicht von uns getestet, andere Linux Betriebssysteme damit zu verwenden. Änderungen können in diesem Fall dazu notwendig sein. Ebenso können bei der Verwendung anderer Cloud Server Hostern eventuell andere Vorgehensweisen notwendig werden.

Wireguard Server mit grafischer Benutzeroberfläche (GUI)

Du hast bei der Bedienung der Wireguard Servers anschließend eine grafische Benutzeroberfläche zur Verfügung und musst keine Programmierkenntnisse mitbringen.

Verbinden kannst Du dich mit den Geräten über die Verwendung der Wireguard Clients. Diese kannst Du in den Appstores, oder auch auf der Wireguard Projektseite finden und laden: [Wireguard Download](#). Es gibt die Clients für viele Gerätearten und Betriebssysteme auch für Windows, MacOS, Android, Apple iOS, Linux usw.

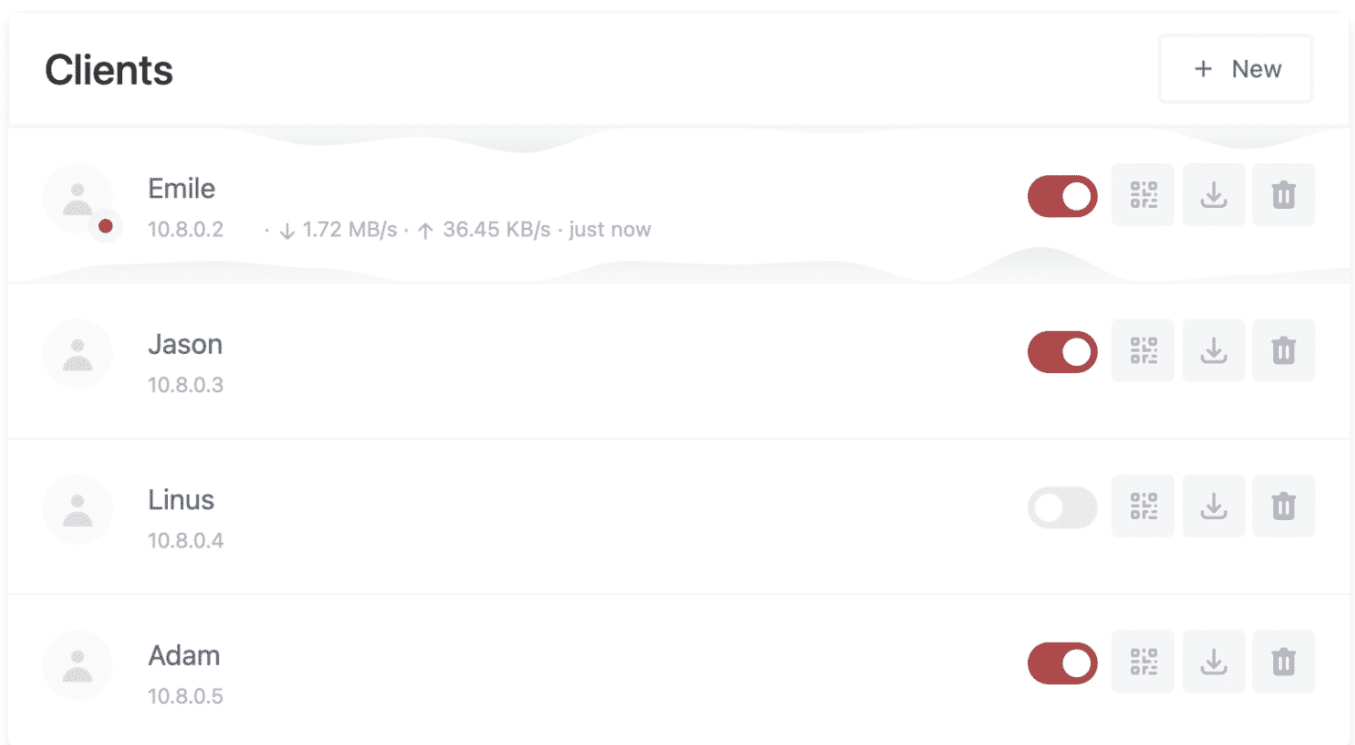
Der Wireguard Client auf Deinem Gerät baut dann einen verschlüsselten VPN Tunnel zum VPN Server auf und fortan werden alle Daten des Gerätes über diese VPN Verbindung gesendet und empfangen. Bei der Nutzung des internets erscheint nun die Server IP Adresse und nicht mehr die des verwendeten und verbunden Gerätes.

Dieser Wireguard Server unterscheidet sich in der Sicherheit und auch Ausführung nicht von den Servern die auch bei den meisten VPN Provider verwendet werden.

Wenn Du Dir nicht ganz sicher bist, dann Folge exakt unserer Anleitung, diese wird dich sicher zum Erfolg führen!

Der ganze Vorgang um einen Wireguard Server selbst betreiben zu können ist sehr einfach und auch für Anfänger der Materie leicht durchführbar.

WireGuard



The screenshot shows the 'Clients' section of a WireGuard web interface. At the top right, there is a '+ New' button. Below it, a list of four clients is displayed, each with a profile icon, name, IP address, status toggle, and action buttons (QR code, download, delete).

Client Name	IP Address	Status	Transfer Rate	Time
Emile	10.8.0.2	On	↓ 1.72 MB/s · ↑ 36.45 KB/s	just now
Jason	10.8.0.3	On		
Linus	10.8.0.4	Off		
Adam	10.8.0.5	On		

Der eigene Wireguard Server ganz ohne Programmierkenntnisse

Vor- und Nachteile eines eigenen Wireguard Servers

Vorteile eines eigenen Wireguard Servers:

- Eine statische IP Adresse verwenden
- Geringe Kosten und keine Bindung
- Wireguard Clients sind für alle geräte verfügbar:
Windows, Mac, Android, iOS, Linux, VPN Router
- Du kannst damit lokale Internetsperren umgehen
- 100% Sicherheit vor Überwachung durch den
Internetanbieter

Nicht geeignet ist ein eigener Wireguard Server für:

- Umgehung von geografischen Sperren bei Streaming Diensten ist nur sehr begrenzt möglich.
- Anonymität und Schutz vor einer Ausforschung sind technisch bedingt nicht gegeben. Daher keine Torrent Nutzung damit.
- KillSwitch und andere sicherheitsrelevante Funktionen sind nicht direkt vorhanden und müssen wenn dann über manuelle Einstellungen am Client umgesetzt werden.

Tipps

- Möchtest Du Wireguard aber mit einem professionellen VPN Dienst verwenden, dann empfehlen wir Dir die „nativen“ [Wireguard VPN Anbieter](#) zu verwenden.
- Möchtest Du Dich noch informieren, was Wireguard ist, dann haben wir einen entsprechenden Artikel der Dir dabei helfen wird: [Was ist Wireguard?](#)

Wir helfen darüber hinaus mit vielen weiteren Berichten zu diesem Thema, verwende dazu einfach unsere Suche auf der Webseite.

Vorbereitungen für den eigenen VPN Server

Es sind nur wenige Dinge notwendig und Du kannst den eigenen Server verwenden. Diese Anleitung ist so einfach wie möglich gehalten und Du brauchst dazu nur wenige Anwendungen, wie zum Beispiel eine gültige EMail Adresse, eine Rufnummer für den Erhalt von SMS Nachrichten und auf Deinem PC/Mac eine Terminalanwendung.(Windows Konsole, Terminal)

Terminalanwendung

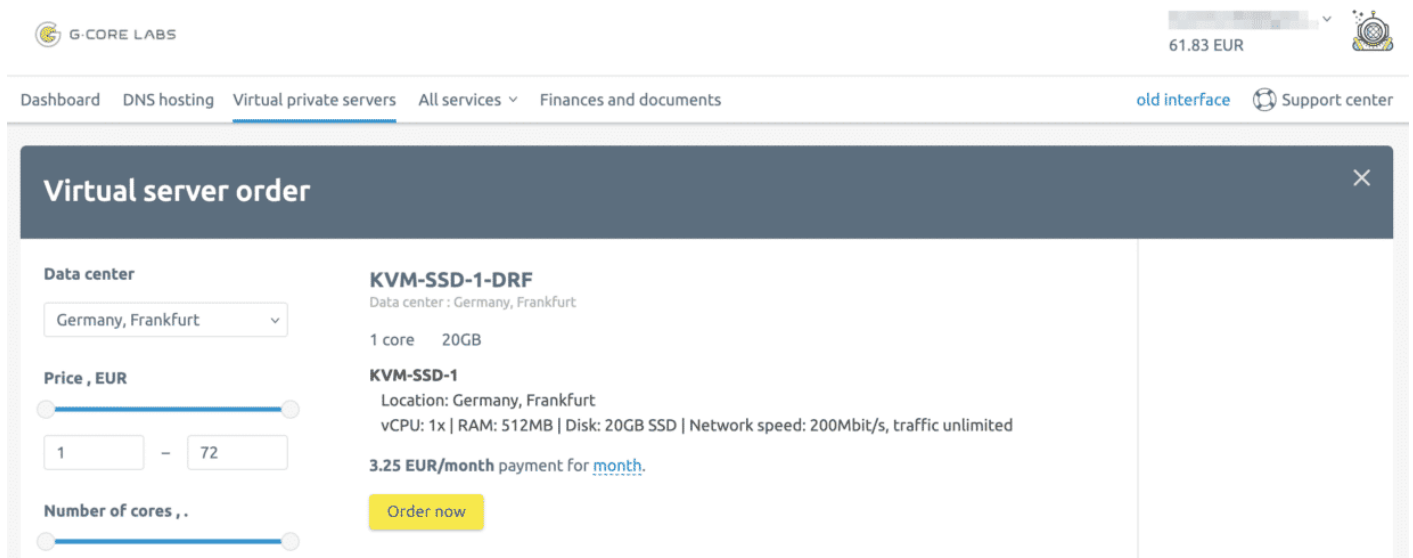
- Die Windows Konsole startest Du einfach mit der Tastenkombination Windowstaste+R öffnet sich „Ausführen“.
- Auf MacOS findest Du „Terminal“ im Zubehör unter den „Programmen“.

"Terminal" im Zubehör unter den "Programmen"

„Terminal“ im Zubehör unter den „Programmen“

vServer bestellen

Du benötigst auf jeden Fall ein G-Core Labs Konto um die Prozedur umzusetzen. vServer sind dort auf täglicher oder monatlicher Basis PREPAID zu mieten. Die Kosten betragen weniger als €3,25 Euro pro Monat. Inkludiert ist eine statische IPv4 und eine IPv6 Adresse. Die Server haben mehr als 99,99% Betriebslaufzeit und sind nicht nur zuverlässig sondern dank SSD Speichern auch sehr schnell.



The screenshot shows the G-Core Labs website interface for ordering a virtual server. The top navigation bar includes 'Dashboard', 'DNS hosting', 'Virtual private servers', 'All services', and 'Finances and documents'. The user's account balance is shown as 61.83 EUR. The main content area is titled 'Virtual server order' and features a dropdown menu for 'Data center' set to 'Germany, Frankfurt'. Below this, there are sliders for 'Price, EUR' (set to 1) and 'Number of cores' (set to 72). The selected server plan is 'KVM-SSD-1-DRF' with specifications: 1 core, 20GB RAM, 1 vCPU, 512MB RAM, 20GB SSD, and 200Mbit/s network speed. The price is 3.25 EUR/month. A yellow 'Order now' button is visible.

GCore Labs Virtuelle Server bestellen

G-Core Labs Konto erstellen

[Videoanleitung: Wireguard Server Installation](#)

Detaillierte Schritt für Schritt Anleitung

Es ist ganz einfach unserer Anleitung zu folgen. Verwende die in der Anleitung vorhandenen Links um zu den richtigen Ressourcen zu springen.

Anleitung: Wireguard Server Installation

In dieser Anleitung zeigen wir Dir Schritt für schritt, wie Du einen eigenen Wireguard Server erstellen kannst und Geräte die Du damit verbinden möchtest verwaltest. Die Anleitung enthält Beispieldaten die Du durch Deine eigenen daten des vServers den Du erstellen wirst austauschen solltest. (IP Adresse des Server und Deine Passwörter). Jeder andere Schritt sollte einfach umzusetzen sein.

1 *G-Core Labs Konto erstellen*

Erstelle ein G-Core Labs Konto indem Du diesem Link folgst: [G-Core Labs](#)

2 *Erstelle einen virtuellen Server*

Im Kundenbereich kannst Du einen „virtuellen Server erstellen“. Das führt Dich zu dieser Seite: [vServer erstellen](#)

3

Standort und Größe wählen

Wähle den für die passenden Standort und die Größe des vServers aus. Du benötigst die folgenden minimum Anforderungen:

- 512 MB Ram
- 1 Core
- mindestens 5GB SSD Speicher

Info: Du brauchst im weiteren Schritt keine zusätzliche IP Adresse auswählen. Eine ist IPv4 und eine IPv6 ist immer inkludiert.

Du kannst in dem folgenden Schritt auch die Zahlungsweise auswählen.

4

Bestellvorgang

Die Konfiguration des Servers sollte auf „Ubuntu 20.xx“ geändert werden. Bei IOv4 und IPv6 braucht es keine Änderungen. Es muss auch keine Domain die statt der IP Adresse verwendet werden kann eingegeben werden.


**5** *Warte bis der Server aktiv ist*

Nun wird der Server erstellt. Dieser Vorgang kann bis zu 10 Minuten dauern. Setze den Vorgang fort, wenn der Server mit „Active“ gekennzeichnet ist.


**6** *Server Konfigurationsdaten*

Du kannst nun auch die Server Konfigurationsdaten einsehen. Du wirst im Anschluß der nächsten Schritte, die IP Adresse des Server als auch das Passwort benötigen.



7

Öffne ein Terminalfenster

Du solltest nun ein Terminalfenster öffnen. In diesem gibst Du ein:

```
ssh root@89.43.107.97
```

Bestätige mit ENTER

Die IP Adresse stammt aus den Serverkonfigurationen die Du bei G-Core im Kundenbereich findest aus dem Schritt 6. Der Benutzername ist immer „root“

8

Anmeldung bestätigen

In diesem Schritt musst Du bestätigen, dass Du die SSH Verbindung aufbauen möchtest. Gib dazu

```
yes
```

ein und bestätige mit ENTER

9

Passwort eingeben

Nun musst Du das Passwort, das Du in der Serververwaltung angezeigt bekommst an dieser Stelle eingeben. Ich empfehle es langsam einzutippen und nicht rein zu kopieren. Danach bestätigst Du mit ENTER

10

Server Daten aktualisieren

In diesem Schritt leitest du den Server an, die neuesten Verweise zu aktualisieren. dazu gibst Du

```
apt-get update
```

ein und bestätigst mit ENTER

11

Server aktualisieren

In diesem Schritt wirst Du nach Updates für den Server suchen. Gib dazu

```
apt-get upgrade
```

ein und bestätige mit ENTER

12 *Docker installieren*

In diesem schritt wirst Du „Docker“ installieren. Gib dazu

```
sudo apt-get install docker.io
```

ein und bestätige mit ENTER

13 *Größe bestätigen*

Du musst nun bestätigen, dass Du die Daten von Docker auf Deinen Server laden willst. Gib dazu

```
Y
```

ein und bestätige mit ENTER

14

Wireguard Server installieren

In diesem Schritt installierst Du den Wireguard Server mit einer grafischen Benutzeroberfläche auf den Server.

Gib dazu die folgenden Befehle ein (als ganzes kopieren und einfügen):

```
docker run -d \  
  --name=wg-easy \  
  -e WG_HOST=89.43.107.97 \  
  -e PASSWORD=DeinPasswort \  
  -v ~/.wg-easy:/etc/wireguard \  
  -p 51820:51820/udp \  
  -p 51821:51821/tcp \  
  --cap-add=NET_ADMIN \  
  --cap-add=SYS_MODULE \  
  --sysctl="net.ipv4.conf.all.src_valid_mark=1" \  
  --sysctl="net.ipv4.ip_forward=1" \  
  --restart unless-stopped \  
  weejewel/wg-easy
```

Die IP Adresse (**89.43.107.97**) und das **DenPasswort** sollte passend zu Deinen Daten aus dem Schritt „6“ ausgetauscht werden.

15

Wireguard Verwaltung im Browser

Nun kannst Du in einem Browser die Wireguard Server Verwaltung öffnen. Gib dazu die IP Adresse Deines Server und den Port 51821 getrennt durch ein „:“ Zeichen ein.

Beispiel: „89.43.107.97:51821?“
und bestätige mit ENTER

Die Verwaltung wird angezeigt.

Du kannst nun Das selbst gewählte Passwort, im Schritt12 ersichtlich“ eingeben und bestätigen.



16 *Client Verwaltung*

Du kannst nun neue Client Zugangsdaten erstellen für Deinen Wireguard Server



17 *Namen des Client angeben*

Du solltest einen eindeutigen und selbstgewählten Namen für die einzelnen Clients die. Du mit dem Wireguard Server verwenden möchtest angeben,

18

Verwaltung der Clients

Di kannst die Konfigurationen für die einzelnen Clients als QR Code anzeigen lassen oder auf Dein Gerät laden. Ebenso kannst Du den Client Sperren oder erneut löschen.

Anpassungen für erfahrene Nutzer

Erfahrene Nutzer können auch weitere Einstellungen am VPN Server vornehmen. Wir setzen hier Ubuntu ein, es kann auch zu notwendigen Anpassungen bei anderen Betriebssystemen wie Debian usw kommen. Diese Installation mit Docker lässt sich auch auf raspberry Pi mit ausreichender Leistung verwenden, wir haben dies allerdings nicht getestet.

Konfigurationen anpassen

Die Konfigurationsdaten des Server werden unter „~/wg-easy“ gespeichert. (config.js) Dort können diese auch später jederzeit angepasst werden.

Optionen bei der Installation

Diese Optionen können durch Setzen von Umgebungsvariablen mit -e KEY="VALUE" im Befehl docker run konfiguriert werden.

Env	Standard	Beispiel	Beschreibung
PASSWORD	–	DeinPasswort	Wenn diese Option aktiviert ist, ist bei der Anmeldung bei der Web-Benutzeroberfläche ein Passwort erforderlich.
WG_HOST	–	vpn.myserver.com	Der öffentliche Hostname Ihres VPN-Servers.
WG_PORT	51820	1234	Der öffentliche UDP-Port Ihres VPN-Servers. WireGuard wird innerhalb des Docker-Containers immer auf 51820 lauschen.
WG_MTU	null	1420	Die MTU, die die Clients verwenden werden. Der Server verwendet die Standard-WG-MTU.
WG_PERSISTENT_KEEP_ALIVE	0	25	Wert in Sekunden, um die „Verbindung“ offen zu halten.
WG_DEFAULT_ADDRESS	10.8.0.x	10.6.0.x	IP-Adressbereich des Clients.
WG_DEFAULT_DNS	1.1.1.1	8.8.8.8	DNS-Server, den die Clients verwenden.
WG_ALLOWED_IPS	0.0.0.0/0, ::/0	192.168.15.0/24, 10.0.1.0/24	Erlaubte IPs, die die Clients verwenden.
WG_POST_UP	...	iptables ...	siehe config.js für den Standardwert.
WG_POST_DOWN	...	iptables ...	siehe config.js für den Standardwert.

Updating

Um auf die neueste Version zu aktualisieren, führen Sie einfach aus:

- `docker stop wg-easy`
- `docker rm wg-easy`
- `docker pull weejewel/wg-easy`

Und führen Sie dann den obigen Befehl „`docker run -d \ ...`“ erneut aus.

Links:

<https://www.edvpfau.de/wireguard-netzwerk-mit-routing-einrichten/>

<https://unix.stackexchange.com/questions/689975/mount-smb-share-after-wireguard-with-fstab-or-systemd>

<https://www.wireguard.com/quickstart/>

DS-Lite IPv6 Tunnel – Port Forwarding – NAT – WireGuard VPN

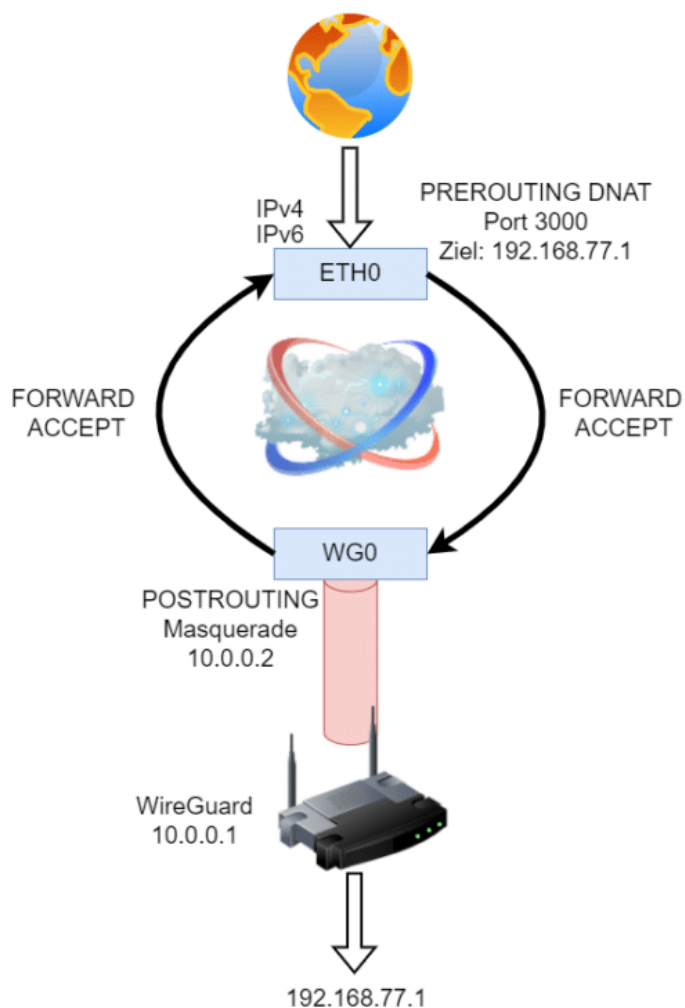


NAT IPv6 Port Forward Cloud Server

Viele Internetnutzer haben eine **Dual Stack** Internetverbindung mit einer nicht eigenen **IPv4 und/oder IPv6** Adresse. Dabei hat man schnell das Problem wenn man auf seine eigenen Dienst zugreifen möchte, keine Verbindung zu Stande kommt. Die IPv4 und die IPv6 Verbindung wechselt

eventuell oder die IPv4 ist eine NAT Adresse usw. Es gibt also viele Hürden wieso man nicht an seine Dienst immer dran kommt.

Im besten Fall sucht man sich also einen Punkt im Internet der gut erreichbar ist und der die Anfragen an deine Dienste weiterleitet. Hier in diesem Beispiel mieten wir uns einen Cloud Server mit einer eigenen IPv4 & IPv6 Adresse. Somit sind die Dienste immer über beide Internet Protokolle erreichbar.



Lösungsbeschreibung

Du brauchst folgendes

- **Cloud Server** -> z.B. Hetzner Cloud (kleinster Server) [HIER 20€ STARTGUTHABEN](#)
- **VPN Gateway** -> z.B. pfSense // [Hardware Appliance](#)

Es wird zwischen dem Cloud Server und deinem VPN Gateway ein Wireguard VPN Tunnel aufgebaut. Diese Verbindung kann über IPv4 oder IPv6 aufgebaut werden. Da diese Verbindung sowieso durch deine pfSense initiiert wird, ist es egal ob v4 oder v6.

Auf dem Cloud Server wird die unten stehende WireGuard Konfiguration genommen und somit auch die 4 Wunderregeln für das DNAT (Destination NAT).

```
[Interface]
Address = 10.0.0.2/24
ListenPort = 51820
PrivateKey = <YOUR-PRIVATE-KEY>
PostUp = iptables -A FORWARD -i eth0 -o wg0 -j ACCEPT
PostUp = iptables -A FORWARD -i wg0 -o eth0 -j ACCEPT
PostUp = iptables -t nat -A PREROUTING -i eth0 -p tcp -m multiport --dport 3000 -j DNAT --to 192.168.77.1
PostUp = iptables -t nat -A POSTROUTING -o wg0 -j MASQUERADE

PostDown = iptables -D FORWARD -i eth0 -o wg0 -j ACCEPT
PostDown = iptables -D FORWARD -i wg0 -o eth0 -j ACCEPT
PostDown = iptables -t nat -D PREROUTING -i eth0 -p tcp -m multiport --dport 3000 -j DNAT --to 192.168.77.1
PostDown = iptables -t nat -D POSTROUTING -o wg0 -j MASQUERADE

[Peer]
PublicKey = <YOUR-PUBLIC-KEY>
AllowedIPs = 192.168.77.0/24, 10.0.0.0/24
```

Mit dieser Konfiguration wird der Port 3000 eingehend auf dem Server mittels DNAT auf die Zieladresse 192.168.77.1 weitergeleitet. (

PREROUTING)

Mit dem **PORTROUTING** wird dem Daten-Paket die IP-Adresse des Wireguard Tunnel aufgezwungen.

Die beiden **FORWARD** Regeln geben dem System die Freigabe Pakete zwischen beiden Interfaces zu verschieben.

Natürlich kann die besonders wichtige Regel „PREROUTING“ so angepasst werden das diese für euch passt. Man kann auch UDP Freigaben oder mehrere Ports gleichzeitig freigeben.

mit

```
iptables -I DOCKER-USER -i eth0 ! -s 127.0.0.1 -p tcp --dport 81 -j DROP
```

Wird die Managementseite des Proxys von außen gesperrt.

Alle weitere Informationen natürlich wie immer im Video.

[Video](#)

Ansichten: 13.909

IPTABLES

1. Overview

When working on a Linux-based server or firewall, understanding how to manage and filter network traffic is crucial. One of the most powerful tools available for this purpose is iptables. It allows us to define rules that control the flow of traffic through network ports, ensuring that only legitimate and authorized traffic can reach our system.

In this article, we'll go through how to use iptables to manage multiple ports efficiently. We'll begin by covering the theoretical aspects of iptables and then demonstrate practical code examples to illustrate how we can manipulate port-based traffic filtering.

2. Understanding iptables Basics

Before diving into managing multiple ports, it's important to understand the fundamentals of how iptables works. At its core, iptables operates by organizing traffic filtering into tables, chains, and rules:

- tables: these define the context in which traffic rules are applied, such as filter (for general traffic filtering), nat (for network address translation), and mangle (for packet modification)

- chains: each table has several chains, such as INPUT, OUTPUT, and FORWARD, in which these chains define the direction of traffic (incoming, outgoing, or forwarded)

- rules: these are the individual conditions that define how traffic should be handled, where each rule specifies criteria like the protocol, source, destination, and port

In the context of managing network ports, we're primarily concerned with the INPUT chain, as this controls the traffic coming into our system.

Let's start by adding a rule to allow incoming traffic on a single port, such as port 80 to allow HTTP requests:

```
$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

Next, let's analyze the command above:

- A INPUT: appends the rule to the INPUT chain

- p tcp: specifies the protocol, which is TCP in this case

- dport 80: targets traffic destined for port 80

- j ACCEPT: accepts the matching traffic, allowing it through the firewall

If we run this command and inspect the rules with iptables -L, we'll see the new rule added:

```
$ iptables -L
```

```
target prot opt source destination  
ACCEPT tcp -- anywhere anywhere tcp dpt:http
```

This allows incoming HTTP traffic. In the upcoming sections, we'll explore how to manage multiple ports.

3. Managing Multiple Ports in iptables

Handling a few ports individually is manageable, but what if we need to manage several ports, such as HTTP, HTTPS, and SSH? Instead of creating separate rules for each port, we can use the multiport module to group them into a single rule.

The multiport module in iptables enables us to specify multiple ports in one rule, simplifying configuration and improving readability. For example, let's look at an example where we allow traffic on ports 80 (HTTP), 443 (HTTPS), and 22 (SSH):

```
$ sudo iptables -A INPUT -p tcp -m multiport --dports 80,443,22 -j ACCEPT  
Chain INPUT (policy DROP)  
target prot opt source destination  
ACCEPT tcp -- anywhere anywhere multiport dports http,https,ssh
```

In the above snippet, we've used the `-m multiport` option, which in turn loads the multiport module that consequently enables us to specify multiple ports during our configuration. In addition, `--dports 80,443,22` specifies the list of destination ports (80, 443, and 22).

On the other hand, one may not want to specify specific ports for known protocols. In this particular situation, we can use `--dport 1000:2000` to open up inbound traffic to TCP ports numbered from 1000 to 2000 inclusive.

Now, we have a single rule that allows traffic on all three critical ports. Essentially, using multiport saves time and reduces clutter in our firewall rules.

4. Combining Ports and Protocols

There are situations where we may need to handle both TCP and UDP protocols for different services. For example, DNS commonly uses both UDP and TCP. In essence, DNS uses UDP for quick lookups and TCP for larger queries. In addition, we can add rules to manage these protocols separately, even for the same ports:

```
$ sudo iptables -A INPUT -p tcp -m multiport --dports 80,443 -j ACCEPT  
$ sudo iptables -A INPUT -p udp -m multiport --dports 53 -j ACCEPT  
Chain INPUT (policy DROP)  
target prot opt source destination  
ACCEPT tcp -- anywhere anywhere multiport dports http,https
```

```
ACCEPT  udp -- anywhere      anywhere      udp dpt:domain
```

Here, we can see that the first rule allows HTTP and HTTPS traffic over TCP. However, the second rule allows DNS queries over UDP specifically on port 53. Finally, we can easily see how both protocols are managed with separate rules. This level of control is critical for handling services that operate across different transport layers.

5. Exploring Security Best Practices

In this section, we'll get to understand some of the security best practices to be able to manage your configurations effectively using multiports.

5.1. Restricting Access to Specific IPs on Multiple Ports

In some use cases, we may need to allow traffic on multiple ports but only from specific IP addresses. For instance, we might want to limit access to SSH and web services to a trusted internal network.

To do this, we can combine the multiport module with IP-based filtering. Let's say we only want to allow traffic from 192.168.1.100 to ports 22 (SSH), 80 (HTTP), and 443 (HTTPS).

Let's check out an example illustrating how to meet our inquiry:

```
$ sudo iptables -A INPUT -p tcp -m multiport --dports 22,80,443 -s 192.168.1.100 -j ACCEPT
```

Chain INPUT (policy DROP)

```
target  prot opt source      destination
ACCEPT  tcp  --  192.168.1.100  anywhere      multiport dports ssh,http,https
```

From the output, we can see that the source column is now populated with the IP address value of 192.168.1.100. In this matter, we specify our source and leave the destination without any constraints, as it still shows anywhere. In particular, this restricts traffic on the specified ports to a single trusted IP. Notably, this is a highly effective way to secure sensitive services like SSH or any other protocol.

5.2. Blocking Multiple Ports

Blocking traffic on certain ports is just as important as allowing it. For example, we may want to block access to unused or vulnerable ports like 8080 (alternative HTTP), 8443 (alternative HTTPS), or 3306 (MySQL).

Let's understand how to utilize the iptables options to meet our goal:

```
$ sudo iptables -A INPUT -p tcp -m multiport --dports 8080,8443,3306 -j DROP
```

Chain INPUT (policy DROP)

```
target  prot opt source      destination
```

```
DROP    tcp -- anywhere      anywhere      multiport dports 8080,8443,mysql
```

Here, we're able to utilize the -j DROP option, which is very useful as it silently drops the traffic without sending a response. Wrapping up, the DROP action is useful for completely ignoring traffic on these ports, improving the system's security.

5.3. Saving and Persisting iptables Rules

By default, iptables rules do not persist across reboots. To ensure that our rules are saved and restored on boot, we need to save them manually. On most systems, we can save the rules using iptables-save:

```
$ sudo iptables-save > /etc/iptables/rules.v4
```

```
# Generated by iptables-save
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp -m multiport --dports 80,443,22 -j ACCEPT
COMMIT
```

This command writes the current rule set to a file (/etc/iptables/rules.v4) that will be loaded when the system boots. By saving our rules, we ensure that our configuration persists even after the server restarts.

6. Troubleshooting Common Issues

Even though iptables is powerful, it's easy to make mistakes that can disrupt traffic or leave systems vulnerable. Regularly, common issues include:

- incorrect rule ordering
- missing the multiport module
- unintentionally dropping legitimate traffic

To tackle the issues above, we can use the following command to debug and review our rules in detail:

```
$ sudo iptables -L -v -n
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target  prot opt in  out  source      destination
  100 6000 ACCEPT  tcp  --  *   *   anywhere   anywhere    multiport dports
http,https,ssh
```

In the example above, we utilized three important options that come with the iptables command:

-L: this is essential for verifying the current firewall configuration and ensuring that the desired rules are in place

-v: this option provides valuable information about the effectiveness of the firewall rules, such as the number of packets being blocked or allowed, which can help us identify potential security issues or optimize the rules for performance

-n: this option is often preferred for troubleshooting and debugging firewall issues, as it provides a more precise and easier-to-read representation of the rules, which can help us identify specific IP addresses or port numbers that are being blocked or allowed

This output provides a detailed view of the traffic counters, protocols, and IP addresses involved. We can use it to verify that our rules are working as expected.

7. Conclusion

In this article, we explored how to use iptables to manage multiple ports efficiently. Furthermore, we covered everything from basic rules to handling multiple ports with the multiport module, combining protocols, restricting access by IP, and persisting our configurations.

By mastering these techniques, we can fine-tune network traffic and enhance the security and performance of Linux systems. Whether managing a single server or an entire network, iptables remains an indispensable tool for systems administrators.